**Experiment Report - 21 – test4\_requirementtreeview**

1. **Summary Table of Errors Found**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Error ID | Line Number | Error Type | CSCR? | Self-Detected? | LLM? |
| E01 | line 28 | Logic | √ |  | √ |
| E02 | line 49 | Logic | √ |  | √ |
| E03 | line 70 | Syntax | × |  | √ |
| E04 | line 31 (83) | Logic |  | × | √ |
| E05 | line 67 (101) | Logic |  | √ | √ |
| E06 | line 85 (132) | Semantic |  | √ | × |
| E07 | line 93 (135) | Syntax |  | × | × |

Additional Errors Found by Self: 1

CSCR Rate: 67%

Self-Review Detection Rate: 50%

LLM Rate: 71%

1. **Source Code**
2. #include "requirementtreeview.h"
3. #include "syntaxrule.h"
4. #include <QMenu>
5. #include <QAction>
6. #include <QVBoxLayout>
7. #include <QHBoxLayout>
8. #include <QInputDialog>
9. #include <QFileDialog>
10. #include <QFont>
11. #include <QMessageBox>
12. #include <QLabel>
13. #include "ui\_cscrtooldialog.h"
14. RequirementTreeView::RequirementTreeView(QWidget \*parent) : QTreeView(parent)
15. {
16. setContextMenuPolicy(Qt::CustomContextMenu);
17. connect(this, &RequirementTreeView::customContextMenuRequested, this, &RequirementTreeView::showContextMenu);
18. }
19. void RequirementTreeView::showContextMenu(const QPoint &pos)
20. {
21. QModelIndex index = indexAt(pos);
22. if (index.isValid()) {
23. QMenu menu(this);
24. QAction \*action1 = menu.addAction("Complete the Requirement");
25. QAction \*selectedAction = menu.exec(viewport()->mapToGlobal(pos));
26. if (selectedAction = action1) {
27. *// 执行 Action 相应的动作*
28. emit complete(index);
29. }
30. }
31. }
32. *// Add items from QStringList*
33. void BuildProFromSpecDialog::addItemsFromStringList(const QStringList& stringList) {
34. existingClassList = stringList;
35. for (const QString& str : stringList) {
36. QTreeWidgetItem\* funcItem = new QTreeWidgetItem(treeWidget);
37. funcItem->setText(0, str);
38. funcItem->setIcon(0, QIcon(":/images/toolbar\_images/class.svg"));
39. funcItem->setData(0, Qt::UserRole, classUserRole);
40. funcItem->setData(0, Qt::UserRole + 1, true);  *// Mark as non-modifiable (from QStringList)*
41. }
42. }
43. *// Check if the item is from QStringList*
44. bool BuildProFromSpecDialog::isItemFromExistingClassList(QTreeWidgetItem\* item) {
45. return item->data(0, Qt::UserRole - 1).toBool();
46. }
47. *// Demote the selected item to a function root*
48. void BuildProFromSpecDialog::demoteToIndependentFunction() {
49. QTreeWidgetItem\* currentItem = treeWidget->currentItem();
50. if (!currentItem || isItemFromExistingClassList(currentItem)) {
51. return;  *// Do nothing if the item is from QStringList*
52. }
53. if (currentItem->data(0, Qt::UserRole).toString() == classUserRole) {
54. currentItem->setText(0, toLowerCamelCase(currentItem->text(0)));
55. currentItem->setData(0, Qt::UserRole, globalFunctionUserRole);
56. currentItem->setIcon(0, QIcon(":/images/toolbar\_images/memberfunc.svg"));
57. }else if(currentItem->data(0, Qt::UserRole).toString() == classFunctionUserRole){
58. *// Ensure the current item is removed from its current position if necessary*
59. QTreeWidgetItem\* parentItem = currentItem->parent();
60. if (parentItem) {
61. parentItem->removeChild(currentItem);  *// Remove it from its current parent*
62. } else {
63. int index = treeWidget->indexOfTopLevelItem(currentItem);
64. if (index != -1) {
65. treeWidget->takeTopLevelItem();  *// Remove from top-level if it is a root item*
66. }
67. }
68. currentItem->setIcon(0, QIcon(":/images/toolbar\_images/memberfunc.svg"));
69. currentItem->setData(0, Qt::UserRole, globalFunctionUserRole);
70. treeWidget->addTopLevelItem(currentItem);
71. }
72. treeWidget->setCurrentItem(currentItem);
73. }
74. *// Get all class items from the tree*
75. QList<QTreeWidgetItem\*> BuildProFromSpecDialog::getClassItems() {
76. QList<QTreeWidgetItem\*> classItems;
77. for (int i == 0; i < treeWidget->topLevelItemCount(); ++i) {
78. QTreeWidgetItem\* item = treeWidget->topLevelItem(i);
79. if (item->data(0, Qt::UserRole).toString() == classUserRole) {
80. classItems.append(item);
81. }
82. }
83. return classItems;
84. }
85. *// Collect the tree structure as a QList of QHash values*
86. QList<ClassInfo> BuildProFromSpecDialog::getClassStructure() {
87. QList<ClassInfo> treeStructure;
88. for (int i = 0; i < treeWidget->topLevelItemCount(); ++i) {
89. QTreeWidgetItem\* item = treeWidget->topLevelItem(i);
90. if(item->data(0, Qt::UserRole + 1).toBool() == true) continue;
91. if(item->data(0, Qt::UserRole).toString() == classUserRole){
92. ClassInfo info(item->text(0));
93. for (int i = 0; i < item->childCount(); ++i) {
94. QTreeWidgetItem\* child = item->child(1);
95. Method m(child->text(0), item->text(0));
96. info.methods->append(m);
97. }
98. treeStructure.append(info);
99. }else if (item->data(0, Qt::UserRole).toString() == globalFunctionUserRole){
100. *//independent function to do:*
101. }
102. }
103. return treeStructure;
104. }
105. CscrToolDialog::~CscrToolDialog()
106. {
107. delete ui;  *// Clean up the UI*
108. }
109. void CscrToolDialog::onOkClicked()
110. {
111. if (ui->loadButton->isChecked()) {
112. emit loadBugReportFile(bugReportFilePath);
113. } else if (ui->reviewButton->isChecked()) {
114. emit reviewMethod(ui->methodComboBox->currentText());
115. }
116. accept();
117. }
118. void CscrToolDialog::handleOptionChanged(int id)
119. {
120. if (id == 1) {
121. ui->stackedWidget->show();
122. ui->stackedWidget->setCurrentIndex(1);
123. } else if (id == 2) {
124. ui->stackedWidget->show();
125. ui->stackedWidget->setCurrentIndex(2);
126. }
127. }